Seán Briscoe – 18321243

CSU33012 – Measuring Software Engineering Report

Measuring software engineering

Software engineering is a discipline that involves the detailed approach to constructing, implementing, and maintaining software projects. Software engineering is often done in teams to ensure that the team is working as effectively as possible to achieve their goal. Knowing what software engineering how can we go about measuring it for the purpose of determining different implementations of software engineering respective effectiveness? Regardless of method if a truly comprehensive measures could be created the potential benefits would be incredibly useful to managers of teams or even software engineers themselves. Collected data could be used in a multitude of ways, such as to measure correlations between behaviour and effective work or results, to compare the effects that individuals have on their peers in order to productively create teams and sub-teams. However there is a complication stemming from the grey ethics of the collection of this data, we must ask ourselves as developers of this kind of technology how far we should go to obtain this information versus the opportunity cost of not pressing far enough in the name of efficiency and accuracy.

How can we measure Software Engineering?

When asked how productivity or quality in software engineering can be quantified or compared, software engineers often reply that it is close to impossible and any attempt to, could even hinder or inhibit the development process. The problem we face initially is identifying effective metrics to measure the software developers by, should we take a rigid quantitative approach when measuring, attributing value to lines of code written or time spent working on the project? Developers argue a system like that would encourage inefficiency and lazy code, a fundamental idea in software development is the idea of constant evolution of code in the pursuit of efficiency, so it’s clear a system like this would be ineffective as it would reward slower code. Another metric we could establish is to measure the work of an engineer by attributing the work they have done over a period of extended work in function points, function points referring to the level of functionality an information system provides to a client in numerical form, to apply this to developers as part of a whole project and it is still somewhat inaccurate in my opinion as I know from experience that the amount of functional code produced doesn’t necessarily correlate with the importance or usefulness of it. If we consider a situation where one team member A, designs a system that is core to the success of the project and is consistently used by the customer, another team member B writes more code that provides different functionality to the system that may not be crucial to the project but does provides a lot of functionality, from a functional point standpoint team member B’s code might garner more FP than team member A’s as it provides more functionality but team member A’s might be more important to the overall performance of the project, so there is an unfair imbalance in the functional point system too. The functional points system is closer to what we need but not quite there yet. Another approach could be to look at individuals themselves instead of the team as a unit, but we quickly run in to issues once more. At the least when we have a group working on a project we can look at how much the entire group has created together but if we were to compare group members we are forced to look at them as if they are all performing the same role, which in practice isn’t as fruitful as a dynamic group with clear roles. Often what we find is that some members can play more facilitative roles by aiding other members in the implementation of their work or providing alternative perspectives on problems a member might be having. So once again we are at a loss, however this is the way that technology is progressing towards a future where data and data interpretation driven artificial intelligence are used to maximise efficiency in the work place and it’s easy to see why.

With that being said there are a set of common metrics that many software engineers agree on being the most important for the success of a measuring system. They include:

* Sprint burndown: referring to a representation of work done by a sprint team versus work left to do, often plotted on a graph. This data is important as you can identify problems if sprint teams consistently finish late or identify good work and chemistry amongst a group if they finish consistently early.
* Team velocity: is the amount of software your sprint team actually produces throughout the sprint. This metric is very useful as it indicates when a team has problems as the velocity will fall compared to other sprints. On top of that it also is a great indicator for if changes were successful as a successful change usually results in an increased velocity.
* Throughput: The throughput of a sprint team is the amount of tickets or completed tasks that the sprint team did throughout a sprint. This is an important metric as the average throughput can be compared to the workload to determine if the workload is too big.
* Cycle time: refers to the time between when work is started on a task usually a ticket, and its completion. We can us this metric to determine if time is being wasted during the sprint as there would be an expected cycle time and also team velocity could be compared to see if time was being wasted.
* Lead time: The time it takes the team to implement a feature since its conception. Similar to cycle time and velocity to determine sprint speed.

The idea of measuring productivity amongst developers is very important and in the eyes of the business or company owner very lucrative. In almost all workplace environments employees’ performance is tracked and evaluated to maintain an intense and competitive work environment while simultaneously keeping tabs on worker performance to maintain standards. Anything from tracking sales from floor staff to tracking sportspeople’s statistics to identify areas in need of improvement. This level of data collection is vital to keep effective teams running smoothly in big companies especially in software engineering where the progression of technology involved and potential for individuals to have major impact is so great, clearly this has been identified as even big companies like Microsoft and Apple collect data on developer performance to maintain standards.

What Platforms can be used to gather and process data?

The big question we now must answer is how we will go about measuring software engineering using tools and frameworks to gather, process and eventually prepare the data in a useful format. The first solution we can look at is the use of a personal software process or PSP which is a framework that helps software engineers looking to improve their performance. The personal software process itself is made up of techniques methods and other structures that aim to help software engineers improve particularly in areas of time management, organisation and code quality, usually in the vision of the company the developer works for. The use of this is simple, using the framework of PSP we can see how far along the and how well a developer has progressed we then collect this data to use later. What is great about the personal software process framework is that there are four clear levels to it, so it is quite concise when it comes to measuring progression along as well as spotting areas where improvement is required. Another tool that is used to track developer data is Microsoft’s workplace analytics which in their words “uses data from everyday work in Microsoft 365 to provide a dynamic view into collaboration patterns, empowering organizations to act with agility, improve employee experiences, and sharpen customer focus.”. In practice through the dashboard we can monitor several metrics of our choice depending on hours worked to see how developers are spending their time. Its used to determine how well developers are working with their teams also as it can track meeting as a metric also, if we choose to go deeper into it workplace analytics can actually determine the effectiveness of meetings and networks using patterns and intelligent metrics. The use of workspace is a more standardised approach to measuring productivity as it can be used to keep track of business teams as well as teams of software engineers however, so in terms of monitoring the effectiveness as a team it works well but in terms of the quality of the code it is less effective. Another tool that can be used by software engineers is Flow made by Pluralsight. Flow is a tool designed to monitor software engineers specifically as it deals with git projects using intelligent metrics to monitor git projects and provide useful metrics to maintain data collections from your team for their code. Flow is dynamically designed tracking employees commits on new work vs older work and also allowing the user to manage the project with more control as it provides all pull requests in one place where they can be interacted with simultaneously, this gives the user or admin the ability to see where disagreements and uncertainty in commits are coming which of course pin points the problem area or person. Flow contains framework to handle interpersonal data also by tracking the level of interaction between members in terms of commits and pull requests, this has obvious implications for the team as if there is a lack of communication it will be clearly outlined by Flow. Flow is a fantastic tool for measuring git projects as well as the engineers working on them by virtue of the fact that it provides such a sheer amount of data to be manipulated, interpreted and referred to as the user wishes. Waydev is another tool used to track and collect data on git projects. Waydev is similar to Flow in that it maintains data concerning git projects, but its focus is an automated system that tracks everything. Since it is highly automated the data is extremely well kept and quick, Waydev also follows a more agile approach to software engineering maintaining sprint data on top of general git data which is a great feature as it gives the user the ability to compare developer progress in this sprint to previous sprints which in turn gives a more complete overview of a developers capabilities in various areas versus others. Waydev also includes a clean user interface when combined with the tables and charts it plots collected data on the user experience is super easy to follow. It also provides a developer summary that displays all data on a single engineer, all potential problems and bottlenecks that are potential inhibiting the engineer’s ability to work effectively. These summaries can also be compared to each other in order to maintain a certain standard of work amongst a team. In general, Waydev is a very modern software employing automation heavily to efficiently provide a high level of accuracy.

What algorithms can we use?

To effectively use and work with the data collected we need to implement intelligent algorithms that interpret the data and return to us the information that is relevant and easy to read. Relevant information is generally made up of mostly of the type of work the engineer is doing in terms of quality and quantity. Some implementations of this type of work involve using the time of commits and time between commits to establish an estimate for how much an individual engineer is coding for, i.e. testing the quantity. Its important to note that using this method avoids a lot of possible infringements of privacy to achieve a similar result, meaning that more accurate answer to the quantity question could have been achieved by having the engineer log everything they or follow them on cctv, but at the risk of invading their privacy using this method achieves effectively the same result as the estimation will not be completely accurate but any problems that would have been identified using the cctv method will also be identified here. To evaluate the quality of the work of the engineer the source code itself is looked at as well as the commits. It goes without saying that a dataset of all commits and the code itself are required for this implementation to work. This principle along with the information itself are passed into a neural network that identifies patterns, other information is also included such as day and time settings which prevent the network from expecting all engineers to code all night etc. The key to modern day processing of data in terms of software engineering productivity is now machine learning and to a degree artificial intelligence. In the case of productivity as in many other cases using machine learning to match patterns in big datasets to evaluate what is a good level of productivity or at least the desired level productivity. For a more in-depth look we can look at how Pluralsight’s Flow which uses over 100 metrics to measure the productivity of the of their clients’ engineers which are broken down into three categories: lines of code, pull requests, and commits which are then assigned to various metrics using machine learning to decide how the engineers end up scoring on the various metrics that Flow provides. According to the CEO of Pluralsight the metrics: lead time, churn, impact, active days, and efficiency with three of the 5 being taken from the lines of code category. In terms of the biggest tech companies where code commits can be upwards of hundreds or thousands of times a day described by their delivery velocity, where much of this can be attributed to implementing both artificial intelligence and machine learning to search for patterns and commonalities in developer practices in order to rapidly maintain accurate levels of development reports.

Is this Ethical?

Possibly the most important question to answer regarding this topic is: is all of this ethically sound? In addition, where do we draw the line in terms of the ethics? In terms of the law THE General Data Protection Regulation or GDPR monitors the protection of data and includes several rights afforded to workers in the workplace. These rights protect them against unwanted data collection if they feel its unlawful or inaccurate. Modern companies must be careful when attempting to record data on their employees and in the case of their engineers they must consider more creative ways to gather required data due to the difficulty in measuring software engineering already. With this knowledge companies have come up with ingenious solutions to gathering data on there developers with companies like Steelcase who have created chairs that monitor the developer sitting in them, providing them with advice on posture and advising breaks while also recording information such as heart rate. In terms of the software engineer companies like Timeuler have created tools that allow tech companies to gather data on their engineers. The Timeuler tracker is a device in the form of an eight sided dice that sits on the user’s desk, each side is programmed to have a different task then when the user is doing a task they leave the corresponding side face up, when they change task for example going from programming to answering emails they turn the dice. Timeuler will keep track of how the developer spends their day and sends it to the user’s phone, this data can be used to see if elements of the developer’s day need to be more optimised. Timeuler is an example of ethically sound data collection usable in the software engineering workplace, in my opinion. In terms of ethics I believe that generally all data that can be derived from commits and interactions with the git project is ok to record and vitally important to the growth and productivity in the modern tech industry. However, when it comes to the developer themselves its more difficult to judge, data should not be collected outside the workplace as it interferes with GDPR rights and even in the workplace we must be vigilant and get creative. As mentioned before there are ways to collect data that don’t risk GDPR infringements such as using commit times and times since last commit values to estimate time spent coding instead of constantly monitoring of developers.

Another aspect of the ethical dilemma of measuring software engineering aside from data collection is how the data is used. Data collected can be used effectively and ethically simultaneously as many companies currently do but its important to be cautious when it comes to using machine learning and artificial intelligence that data collected is used in a positive justifiable manner. Artificial intelligence will point out and advise people in managerial positions on developer productivity in terms of metrics that we provide, as a result if the metrics provided are not completely effective and useful then incorrect advise might be provided by the artificially intelligent system. There is a view that artificial intelligence is perfect in terms of accuracy and is rarely if ever wrong, is can be a misconception when the data provided is incorrectly gathered or incorrect the artificially intelligent system will provide incorrect outputs as a result, so caution must be practiced. There are numerous examples of where artificial intelligence interpreting data has gone wrong so as we move into a time where it is used more frequently, we need to maintain a level of caution. However, I am in favour of using artificial intelligence to optimise the way we measure software engineering personally if we maintain an ethical approach, the use of data in the workplace of a software engineer is great for software engineers everywhere as it will lead to a more clear cut way of quantifying and evaluating the value of your work which is results in a more competitive and creative marketplace of ideas as well as tech industry as a whole.

Conclusion:

To conclude there are several potential barriers to instrumenting software engineering productivity such as ethics and the complexity of the task itself. However, many companies are already effectively monitoring this kind of data and the number of companies as well as the amount of data will continue to grow. The value of data manipulation and interpretation in the industry of software engineering is incredibly high due to the results it yields when done correctly.

Sources:

* https://www.youtube.com/watch?v=Dp5\_1QPLps0
* http://2016.msrconf.org/
* https://www.researchgate.net/profile/Jan\_Sauermann2/publication/285356496\_Network\_Effects\_on\_Worker\_Productivity/links/565d91c508ae4988a7bc7397.pdf
* https://wiprodigital.com/2019/08/28/future-of-software-development-ai-in-developer-productivity-part-1/
* https://callminer.com/blog/important-algorithms/
* http://www.hitachi.com/rev/pdf/2015/r2015\_08\_116.pdf
* https://kambria.io/blog/the-7-most-pressing-ethical-issues-in-artificial-intelligence/
* https://link.springer.com/article/10.1007/s10664-005-1290-x
* https://timeular.com/
* https://www.citizensinformation.ie/en/employment/employment\_rights\_and\_conditions/data\_protection\_at\_work/data\_protection\_in\_the\_workplace.html
* https://www.gitclear.com/pluralsight\_gitclear\_pinpoint\_code\_climate\_code\_development\_kpi\_metric\_alternatives
* https://www.youtube.com/watch?v=KKJ6Ec9jYgI + https://semmle.com/assets/papers/measuring-software-development.pdf